FARABI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE



Lecture 2.
MPI Overview, Process Model and Language Bindings

Dr. Bazargul Matkerim, Professor-Researcher

Outline

1. **MPI overview**

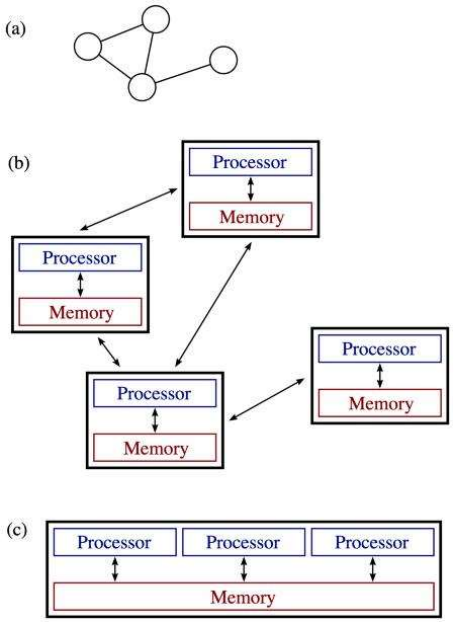
   message-passing paradigm, SPMD, messages

2. **Communication concepts**

   point-to-point, blocking and non-blocking, MPI Forum

3. **Process model and language bindings**

   MPI_Init, communicator, rank, size, MPI_Finalize



Source: Distributed vs parallel, commons.wikimedia.org (CC BY-SA 3.0)

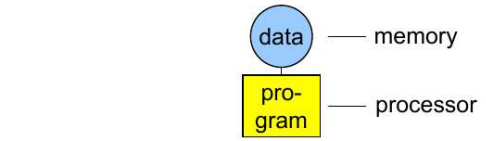
The Message-Passing Programming Paradigm

Each processor runs a **sub-program**
- written in a conventional language, e.g. C or Fortran
- typically the same on each processor (SPMD)

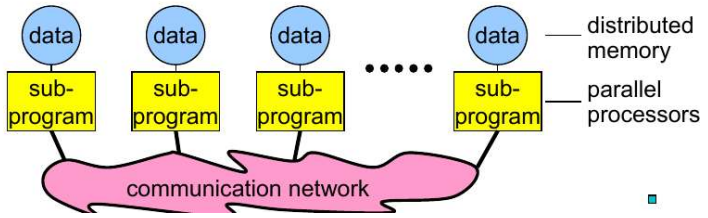
All variables are **private**
- same name, but different locations and different data

Communication via special **send & receive** routines

- **Sequential Programming Paradigm**



- **Message-Passing Programming Paradigm**



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 8

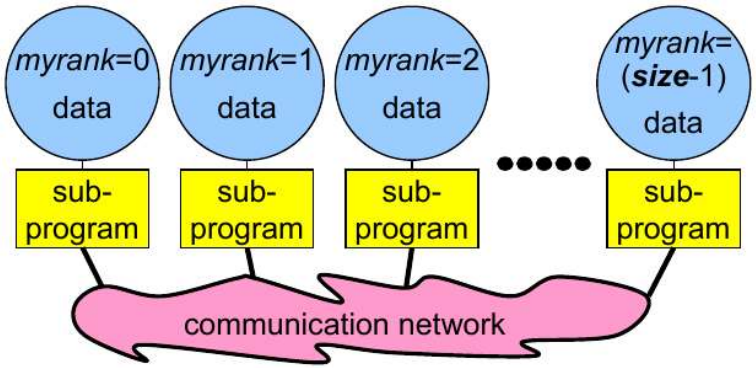
Data and Work Distribution

The value of **myrank** is returned by a special library routine

A system of **size** processes is started by mpirun or mpiexec

All distribution decisions are based on **myrank**

i.e. which process works on which data



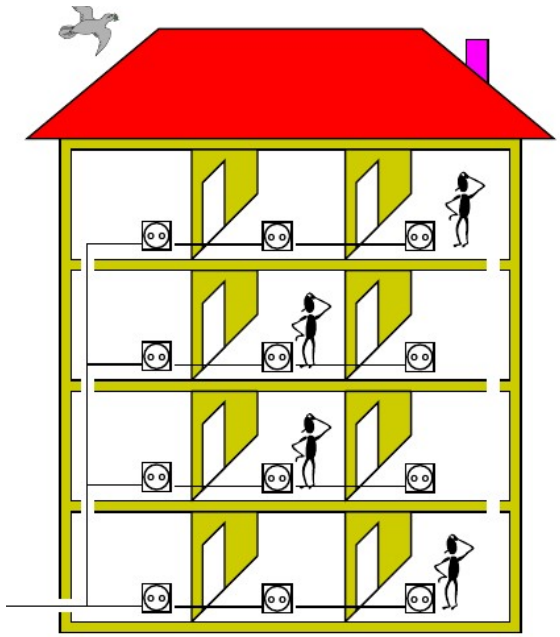
Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 10

Analogy: Electric Installations in Parallel

MPI sub-program = work of one electrician on one floor

Data = the electric installation

MPI communication = real communication
to guarantee that the wires come at the same position through the floor



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 11

What is SPMD?

Single **P**rogram, **M**ultiple **D**ata

The same (sub-)program runs on each processor

MPI also allows **MPMD** (multiple programs)
some vendors may be restricted to SPMD

MPMD can be emulated with SPMD

Emulation of MPMD in C

```
main(int argc, char **argv)
{
  if (myrank < ... /* ocean model */) {
    ocean( /* arguments */ );
  } else {
    weather( /* arguments */ );
  }
}
```

Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 13

Messages

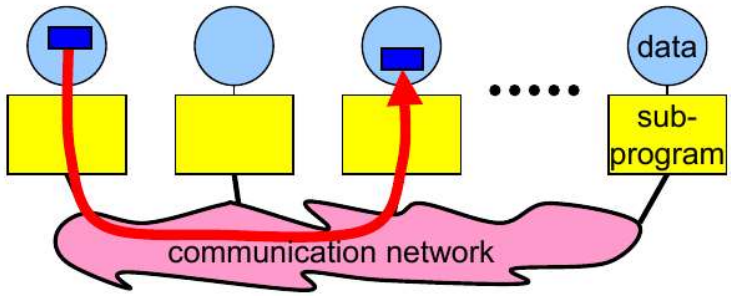
Messages are **packets of data** moving between sub-programs

The message passing system needs:

- sending and receiving process (the ranks)
- source and destination location
- source and destination data type
- source data size and destination buffer size

Addresses in MPI are the **ranks**

All messages must be **received**



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 14

Point-to-Point Communication

Simplest form of message passing

One process sends a message to another

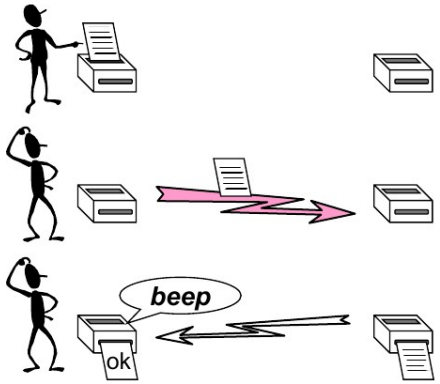
Synchronous send

the sender knows the message is received

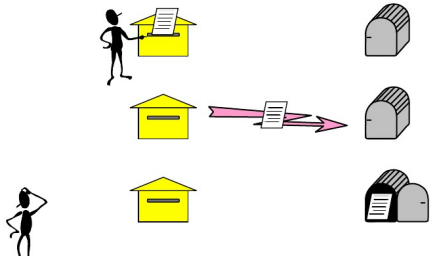
like the beep or okay-sheet of a fax

Buffered = asynchronous send

only know when the message has left



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 19



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 20

Blocking and Non-Blocking Operations

Blocking: returns only when the operation has completed

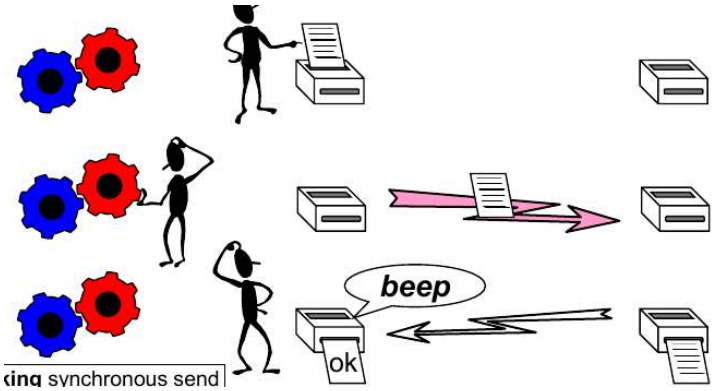
synchronous send blocks until the receive is posted

Non-blocking: returns immediately

later the program must test or wait for completion

Collective communication: several processes

broadcast, reduction, barrier



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 22

MPI Forum and Goals of MPI

MPI-1 Forum: 60 people from 40 organizations
- MPI 1.0: June 1994
- MPI 1.1: June 12, 1995

MPI-2: extensions, July 18, 1997

Goals:
- a message-passing interface
- source-code portability
- efficient implementations
- support for heterogeneous architectures



Source: NASA Pleiades supercomputer, commons.wikimedia.org (public domain)

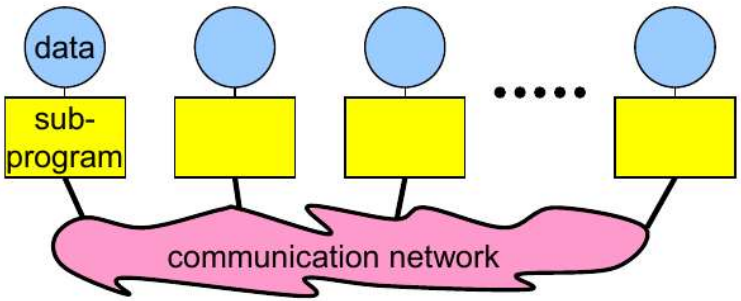
Initializing and Starting MPI

Header: #include <mpi.h>

MPI_Init must be the **first** MPI routine

Start: mpirun -np 4 ./prg
or with MPI-2: mpiexec -n 4 ./prg

Processes exist at least after MPI_Init



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 36

```
#include <mpi.h>
int main(int argc, char **argv)
{
  MPI_Init(&argc, &argv);
  ....
}
```

Communicator, Rank and Size

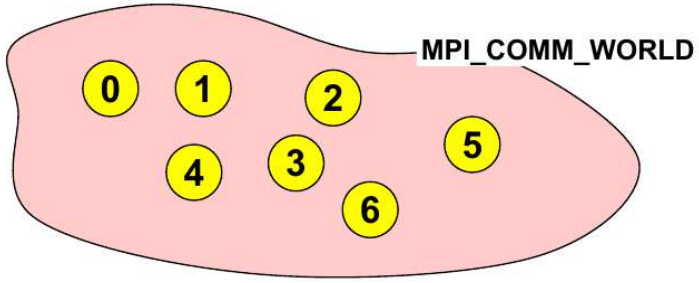
MPI_COMM_WORLD: all processes of the program

Rank: from **0** to **size − 1**

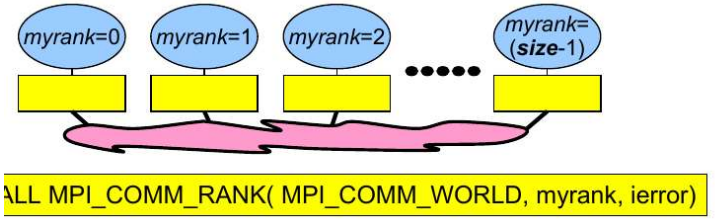
MPI_Finalize() must be called **last**

```
int myrank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
printf("I am %d of %d\n", myrank, size);
MPI_Finalize();
```

Thank you for your attention!



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 37



Source: R. Rabenseifner, HLRS, Introduction to MPI, slide 39